웹응용기술

Node.js를 활용한 Profiler 프로그램 분석

과목	웹응용기술[002]
교수	강영명 교수님
과제명	Node.js를 활용한 Profiler 분석
제출일	2025.06.06
학번	20200854
이름	윤민혁

< 목차 >

< 그림, 표 목차 >

1. 프로그램 개요

본 프로그램은 Node.js를 활용한 Profiler 프로그램 분석으로써, 사용자 입력 또는 데이터베이스에 저장되어 있는 CPU의 Core와 Core별 Task의 수행 시간 데이터를 자동으로 분석하여 웹 기반으로 시각화하는 기능을 제공한다.

사용자는 Profiler로 수집된 데이터를 업로드 할 수 있고, 데이터베이스에 저장된 분석 결과를 Chart.js를 이용한 막대그래프와 파이그래프로 조회할 수 있으며, Task별 및 Core 별 처리 시간에 대한 최소(MIN), 최대(MAX), 평균(AVG) 값을 시각적으로 확인할 수 있다.

본인은 웹응용기술 강의에서 배운 Node.js와 ChatGPT 및 인터넷 검색 등의 도움을 받아 프로그램을 개발했다.

2. 프로그램 수행 절차 분석

2-1. 프로그램 실행 및 접속

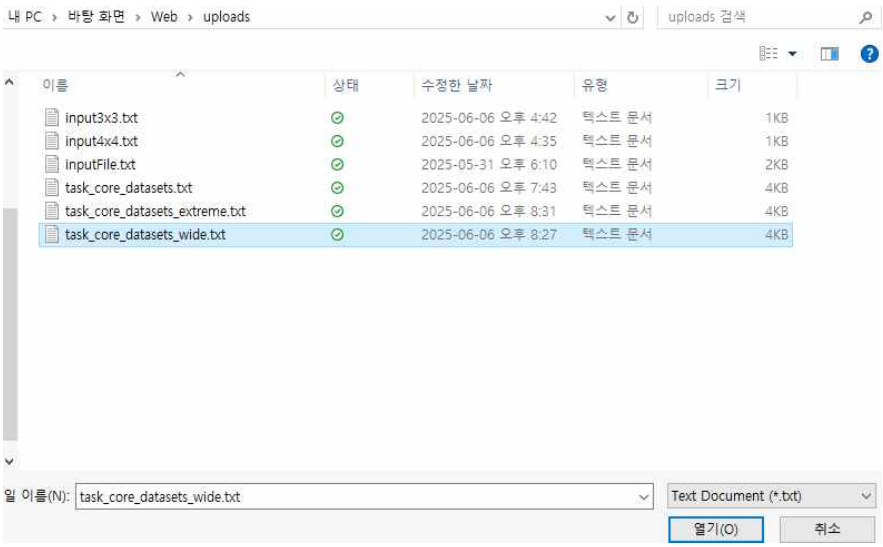
GitHub에서 파일을 다운로드 받은 뒤, 통합 터미널을 열어 “**node app**” 또는 “**node app.js**”를 입력하면, localhost:8081의 서버가 열렸다는 메시지와 데이터베이스 연결 성공 메시지가 출력된다.

이후 인터넷의 **localhost:8081**에 접속하면 웹페이지가 정상 작동하는 것을 알 수 있다.

[그림 1] localhost:8081 접속

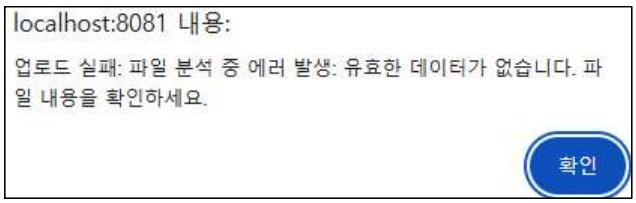
2-2. 데이터 입력

사용자는 상단의 파일 선택 버튼과 업로드 및 분석 버튼을 통해 직접 Profiler로 수집된 txt 파일 형식의 데이터를 웹페이지로 업로드 하거나, 데이터베이스에 미리 저장된 데이터를 선택할 수 있다.



[그림 2] 데이터 입력

task와 core로 지정된 형식의 파일만을 수용하기 때문에, 원하지 않는 데이터가 입력이 되면, 오류 메세지를 출력해 잘못된 파일의 입력을 방지한다.



[그림 3] 잘못된 형식 파일 제한

2-3. 데이터 파싱 및 저장

업로드된 파일은 서버 측(Node.js)에서 파싱되어 **Task 단위와 Core 단위**로 데이터를 분리하여 각각의 구조에 맞게 데이터를 저장한다. 데이터를 분리하여 저장하는 목적은 Task별 분석과 Core별 분석을 독립적으로 수행하여 결과를 도출해내기 위함이다.

처리된 데이터는 다음 두 가지 방식 중 하나로 저장된다.

· **데이터베이스(DB) 저장 방식** : profiler_db(DATABASE) > profilerData(TABLE)에 저장

· **파일 저장 방식** : /uploads/file.txt 형식으로 서버 로컬에 저장

이 데이터는 프로그램 내 app.js 또는 index.html 파일의 주석 처리된 부분에서 직접 수정 가능하도록 구성되어 있어, 프로그래머가 필요에 따라 원하는 방식으로 저장할 수 있도록 설계했다.

[표 1] profilerData 테이블 구조

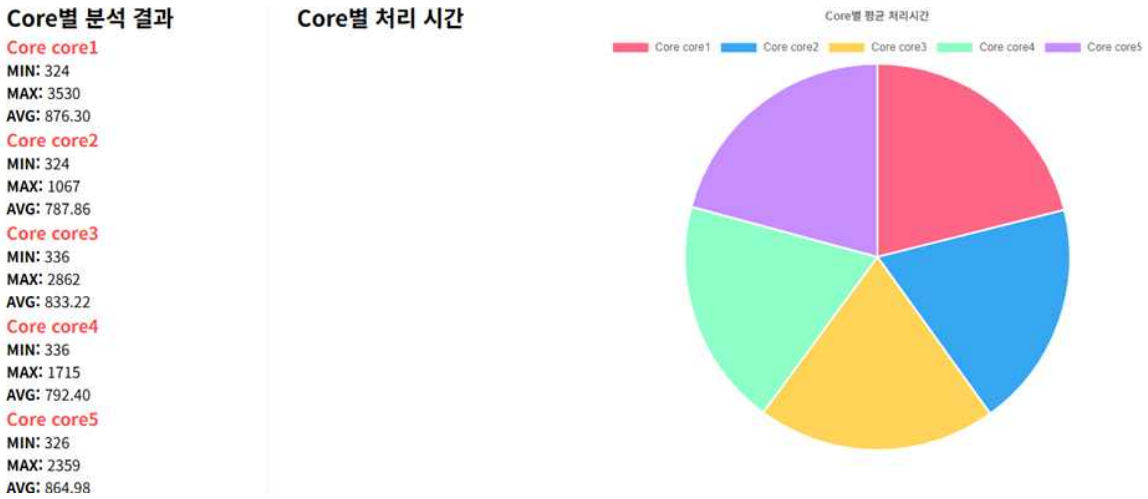
필드명	데이터 타입	제약조건	설명
id	INT	PRIMARY KEY, AUTO_INCREMENT	고유 식별자(자동 증가)
fileName	VARCHAR	NULL 허용	업로드된 파일명 (multer로 랜덤 해시 적용)
core	VARCHAR	NULL 허용	Core 번호
task	VARCHAR	NULL 허용	Task 번호
value	INT	NULL 허용	각 수행 시간 값

2-4. 분석 수행 및 시각화

Core와 Task의 저장된 데이터를 기반으로 수행 시간(MIN, MAX, AVG 등)을 계산한다.

파일은 업로드 즉시 수행시간을 분석하며, 분석 결과는 Chart.js를 사용하여 막대그래프 및 파이그래프로 시각화한다.

[그림 4] Task별 수행시간 비율 막대그래프



[그림 5] Core별 수행시간 비율 파이그래프

사용자는 데이터베이스에 저장된 데이터에 접근하거나 직접 업로드한 파일에 대한 결과를 얻을 수 있다.

2-4. 결과 확인 및 활용

시각화된 결과를 통해 사용자는 CPU의 Core와 Core별 Task의 수행시간을 한 눈에 알아볼 수 있고, 이를 바탕으로 시스템의 성능을 진단하거나 특정 Core, Task의 병목을 파악할 수 있다. 또한 각 그래프에 마우스 오버 시 각 수행시간(MIN, MAX, AVG 등)의 정확한 값을 확인 할 수 있다.

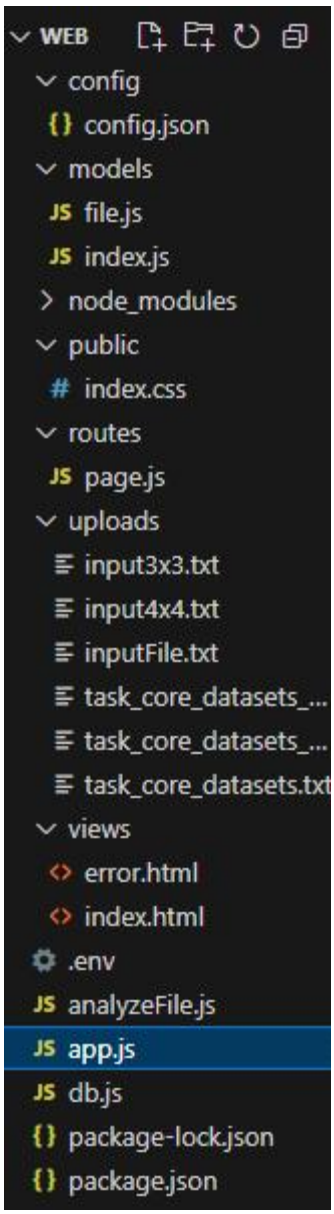
[그림 6] 4x5데이터의 수행결과 및 웹의 최종 형태

위 그림은 [Task x Core]의 4x5 분석 결과이다.

3. 프로그램 기능 분석

3-1. Node.js 폴더 구성

폴더는 루트 폴더인 WEB을 시작으로 여러 기능을 하는 폴더와 파일들로 이루어져있다. 본 프로젝트는 아래 [그림 7]과 같은 디렉터리 구조를 갖고 있다.



[그림 7] Node.js 구조

[표 2] Node.js 디렉터리 구조

항목	설명
config/config.json	DB 정보와 DB 계정 정보 등 저장
models/file.js	Sequelize를 이용해 정의한 데이터 모델로, profilerData 테이블과 매핑되는 구조를 저장
public/index.css	views/index.html의 스타일시트
routes/page.js	/analyzeFile.js의 내부 로직과 연결되는 파일로, Express의 라우터 기능을 이용해 메인페이지 요청/파일업로드 및 분석 처리하는 파일
uploads/	사용자가 업로드한 입력 파일 저장소
views/index.html	사용자가 접근하는 웹 HTML 화면 파일 및 script를 이용해 Core별, Task별 canvas를 생성하고, 분석 결과를 화면에 띄워주는 파일
analyzeFile.js	업로드된 파일을 분석해 Task별 최소, 최대, 평균값을 계산하는 핵심 분석 모듈
app.js	서버의 실행을 담당하는 메인 엔트리 포인트로서, 미들웨어 설정, 파일 업로드 처리, 데이터 분석, 라우팅, 에러 처리까지의 프로그램의 전반적인 핵심 흐름을 구성하는 파일
db.js	Sequelize를 이용해 데이터베이스와 연결을 설정하는 파일

3-2. app.js 주요 기능 분석

app.js는 웹 서버의 핵심 구동 파일이며, Express 기반 웹 애플리케이션의 여러 기능을 담당한다.

- 미들웨어 및 템플릿 설정
- 파일 업로드 및 분석
- 데이터베이스와의 연동
- 라우팅 및 오류 처리
- 서버 실행

```
const express = require('express');
const path = require('path');
const fileUpload = require('express-fileupload');
const session = require('express-session');
const cookieParser = require('cookie-parser');
const nunjucks = require('nunjucks');
const dotenv = require('dotenv');
const morgan = require('morgan');
const multer = require('multer');
```

서버 실행과 기능 구현에 필요한 여러 Node.js 모듈을 불러왔다.

[그림 8] 모듈 불러오기 및 환경 구성

```
const app = express();
app.set('port', process.env.PORT || 8081);
app.set('view engine', 'html');
nunjucks.configure('views', { express: app, watch: true });
```

express로 서버를 초기화 시켜준 뒤, 포트번호를 8081로 설정했다. 또한 넌적스 템플릿 문법을 사용 가능하도록 조치하였다.

[그림 9] 서버 설정 및 템플릿 엔진 구성

```
const { sequelize } = require('./models');
sequelize.sync({ force: false })
  .then(() => console.log('데이터베이스 연결 성공'))
  .catch(err => console.error('데이터베이스 연결 실패:', err));
```

models/index.js를 통해 Sequelize 인스턴스를 불러와 DB를 연결한다.

[그림 10] 데이터베이스 연결(Sequelize)

```
app.post('/upload', upload.single('file'), async (req, res) => {
  ...
  res.json({
    message: '파일 업로드 및 분석 성공 (DB 저장 없음)',
    result: resultArray,
  });
});
```

DB에 저장하지 않고, 분석만 수행했다.

코드 내부에는 파일의 \t와 \n값에 대한 처리 코드도 있다.

또한, DB에 저장하려면, 임의로 주석 처리한 값의 주석을 해제하면 된다.

[그림 11] 파일 업로드 및 분석 처리

```
app.get('/results/:fileId', async (req, res) => {
  ...
  res.json({
    taskData: [...],  // task별 통계
    coreData: [...],  // core별 통계
  });
});
```

DB에서 fileName에 해당하는 값을 기준으로 분석 데이터를 조회해 Task와 Core별 통계를 가져온다.

MIN, MAX, AVG 그룹 별로 계산하여 JSON 응답을 해준다.

[그림 12] 분석 결과 조회 (DB 기반)

```
app.use('/', pageRouter);

app.use((req, res, next) => {
  const error = new Error(`${req.method} ${req.url} 라우터가 없습니다.`);
  error.status = 404;
  next(error);
});

app.use((err, req, res, next) => {
  res.locals.message = err.message;
  res.locals.error = process.env.NODE_ENV !== 'production' ? err : {};
  res.status(err.status || 500);
  res.render('error');
});
```

존재하지 않는 URL 요청에 대해서는 404 오류 처리를 한다.

라우터를 등록하고 오류를 처리해주는 구문이다.

[그림 13] 라우터 등록 및 오류 처리

```
app.listen(app.get('port'), () => {
  console.log("http://localhost:"+app.get('port')+" server open");
});
```

8081포트에서 서버를 실행하고, 콘솔에 주소를 출력해준다.

[그림 14] 서버 실행

3-3. index.html 주요 기능 분석

index.html은 HTML 화면을 구성해주는 것뿐만 아니라, 파일 업로드와 script를 통해 웹 페이지에 여러 기능을 부여했고, Chart.js를 이용해 시각화한 중요한 파일 중 하나다.

· HTML 화면 구성

1. HTML UI를 구성해 파일 업로드 폼을 만들고 POST 요청을 한다.
2. DB 데이터 불러오기 버튼 3가지(5x5, 4x5, 3x3)

· script 구성

1. 파일 업로드 폼에 대한 처리 스크립트
   - 서버에 파일 업로드를 요청하고, 분석 결과를 받아 renderResults, renderChart로 시각화

2. renderResults, renderCoreResults 함수
   - .innerHTML을 통해 ul, li 구조를 만들고 그 안에 Task또는 Core 결과를 출력해준다.

3. renderChart, renderCoreChart 함수
   - Task, Core 차트를 만들어 Task, Core별 통계(MIN, MAX, AVG)를 시각화를 해준다.

4. loadDbData 함수
   - 공통 로드 함수로, 버튼 클릭 시 서버에서 받는 분석 결과를 가져와 여러 오류를 검출한 뒤, 오류 검출에 걸리지 않으면 위에서 가져온 데이터를 renderChart와 renderCoreChart로 시각화한다.

· 전체 흐름 요약

1. 사용자 업로드

- 사용자 파일 업로드 -> 서버 분석 -> JSON 결과를 가져와 화면에 시각화

2. 저장된 DB 불러오기

- 미리 지정해둔 DB의 fileName이 담긴 버튼 클릭 -> 해당 값의 차트 및 분석 결과 출력

4. 소스 코드 (GitHub)

프로그램을 구성하는 소스 코드는 본인의 깃허브에 저장해두었다.

깃허브 주소 : https://github.com/ymh010704/Web_Profiler

5. 참고자료

1. ChatGPT, OpenAI, https://chat.openai.com, (참고일: 2025.05.24.~2025.06.06.)

2. 웹응용기술 강의자료, 성결대학교, 담당교수 : 강영명, 2025-1학기

3. 웨이먼드, ChatGPT로 Chart.js 알아보기 (3): 막대 차트(Bar Chart), Little joys, (2023.06.18.), https://waymond.tistory.com/55,